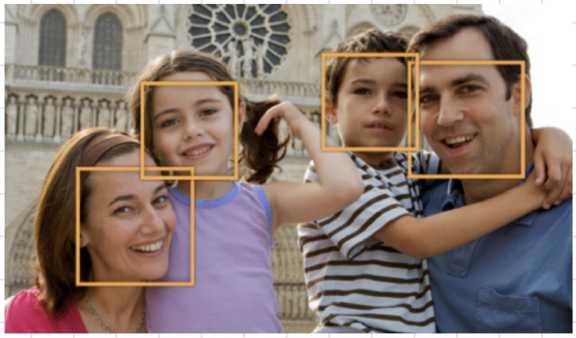
LECTURE 2 : FACE DETECTION



Face detection is part of object detection / recognition, which most of the techniques are based on machine learning.

↓↓

Machine Learning's Common Framework

Training stage
(supervised learning)

Input : data + labels

Output : model parameters

Testing stage :

Input : query data + model parameters

Output : predicted labels

Data = images → transform to Features = Important factors of images

↓↓

Face detection technique : Viola - Jones Face Detection.

Basic components : (1) Features
(2) Classification

Feature Extraction : Image Convolution

» 1D Convolution :

Analog operation : $(f * g)(x) = \int f(u)\, g(x-u)\, du$

Discrete operation : $(f * g)(x) = \sum_{u=-\infty}^{+\infty} f(u)\, g(x-u)$

e.g. :

f = the 1D kernel = | 1 | 2 | 3 | (u: +1 0 -1)

g = the input signal = | 5 | 10 | 7 | 0 | ... | (x: 1 2 3 4 5)

$(f * g)(x=2) = (3 \times 7) + (2 \times 10) + (1 \times 5) = 46$

$(f * g)(x=3) = (3 \times 0) + (7 \times 2) + (1 \times 10) = 24$

$h(x) = (f * g)(x)$ = the output = | 0 | 46 | 24 | ... |

» 2D Convolution :

$$(f * g)(x,y) = \sum_{v=-\infty}^{+\infty} \sum_{u=-\infty}^{+\infty} f(u,v)\, g(x-u, y-v)$$

$f(u,v)$ = the 2D kernel

e.g. :

k =

-1	0	+1
-2	0	+2
-1	0	+1

(u → , v ↓)

I = Input image : $(k * I)(x,y) = \sum_{v} \sum_{u} k(u,v)\, I(x-u, y-v)$

See on the internet {
- sobel operator
- Gaussian kernel / filter
- Laplacian kernel / filter

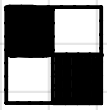
[2] Feature Extraction: Haar-like Features

① Three types of masks:

Type 1:

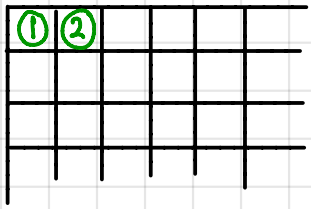
Type 2:

Type 3:



② Convolution with the masks:

I =



e.g: I *

Assuming pixel ① has intensity 100 and pixel ② 150, then:

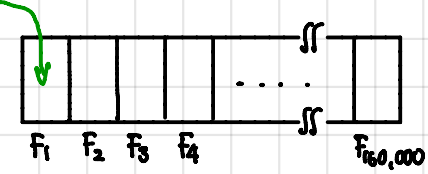
$F = (100 \times -1) + (150 \times +1)$

$= 50$

③ Descriptor:

For an image of size 24 x 24, the convolution of the image with all types of the masks & all possible sizes will generate 160,000 values of F (= features)

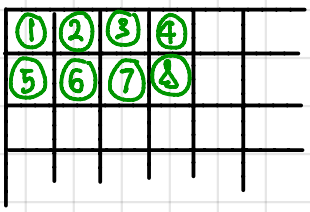
F =



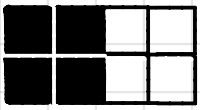
This vector is called a descriptor

Another example:

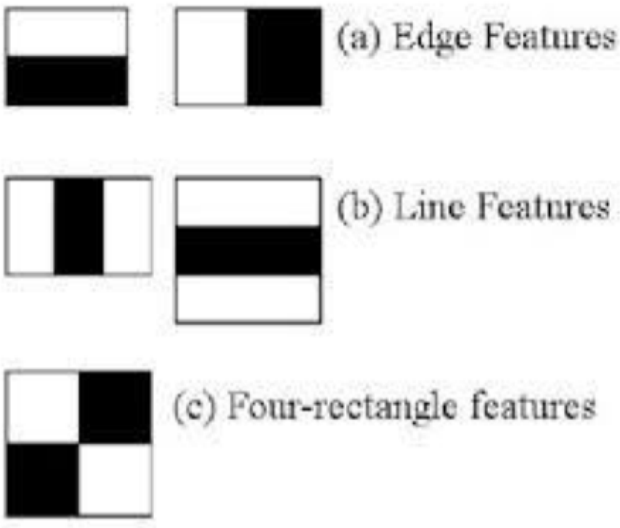
I =

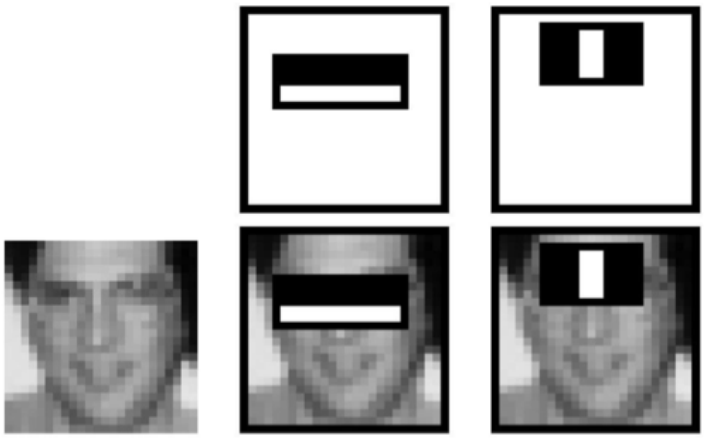


*

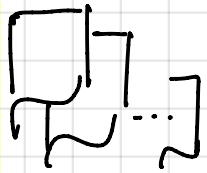


$F = [③ + ④ + ⑦ + ⑧] - [① + ② + ⑤ + ⑥]$





[3] Training Pipeline



⊕ & ⊖ samples

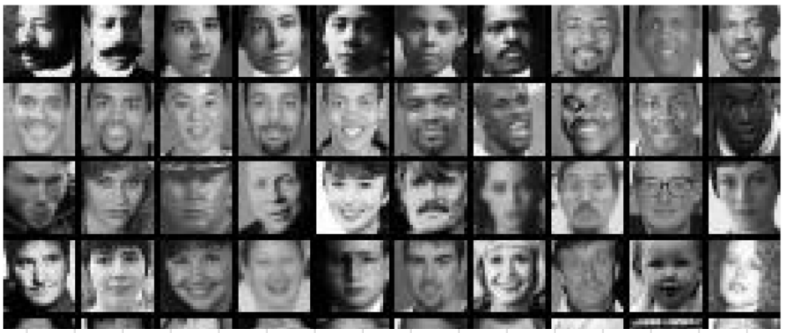
→ Feature Extraction → 10,000 descriptors where each has a length of 160,000 → Classification: AdaBoost → parameters that can classify face from others

⊕ = 5000 face images of size 24x24

⊖ = 5000 non-face images of size 24x24

Hard negative samples

The positive ⊕ samples:



The code for generating 160,000 features from a single 24 x 24 image:

```
const int frameSize = 24;
const int features = 5;
// All five feature types:
const int feature[features][2] = {{2,1}, {1,2}, {3,1}, {1,3}, {2,2}};

int count = 0;
// Each feature:
for (int i = 0; i < features; i++) {
    int sizeX = feature[i][0];
    int sizeY = feature[i][1];
    // Each position:
    for (int x = 0; x <= frameSize-sizeX; x++) {
        for (int y = 0; y <= frameSize-sizeY; y++) {
            // Each size fitting within the frameSize:
            for (int width = sizeX; width <= frameSize-x; width+=sizeX) {
                for (int height = sizeY; height <= frameSize-y; height+=sizeY) {
                    count++;
                }
            }
        }
    }
}
```

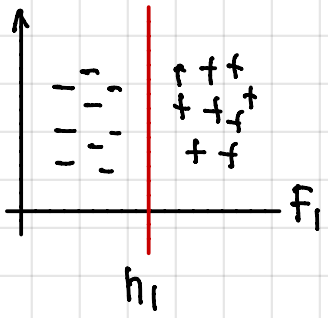
[4] AdaBoost

Goal: to find functions that separate positive & negative samples

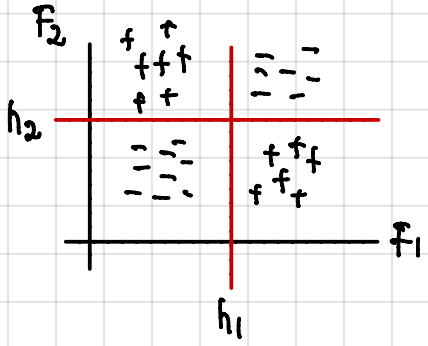
Basic idea: to have a strong classifier (or function) composed of a number of weak classifiers.

Examples:

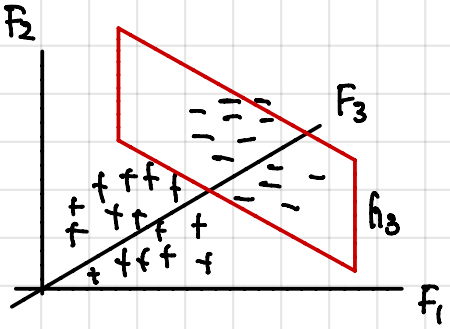
10 data



20 data

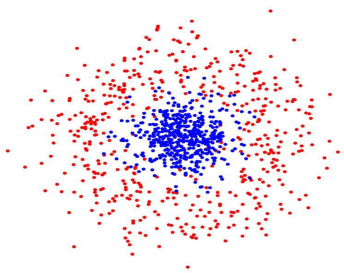


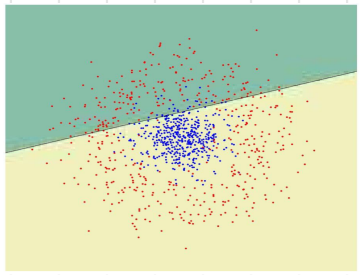
30 data

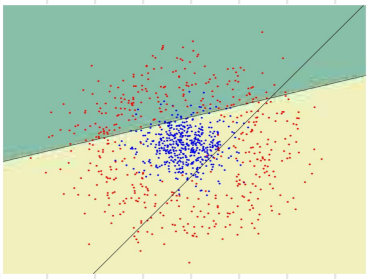


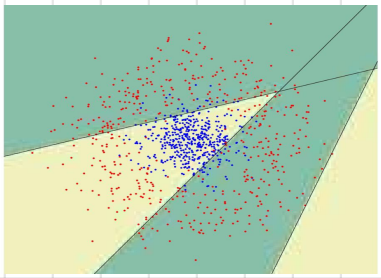
1. $h_1, h_2, h_3, \ldots$ are called weak classifiers that can form a strong classifier.
2. In our case, we have 160,000 features, meaning we have 160,000 dimensions of space.

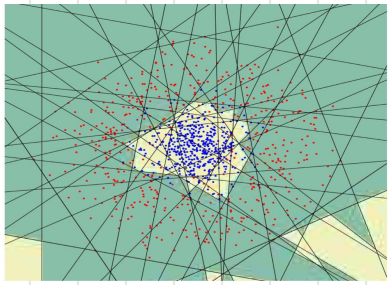
Illustration of the general AdaBoost algorithm:

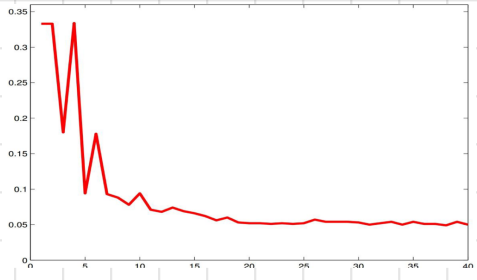












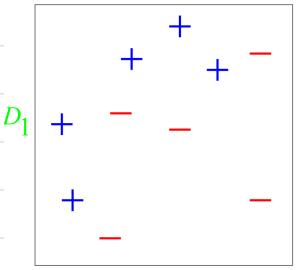
[5] AdaBoost: Weights

AdaBoost has important properties:

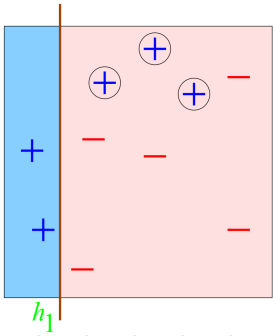
1. Iterative in nature
2. In each iteration, it generates a weak classifier
3. In the next iteration, it focuses on samples that have more weights.

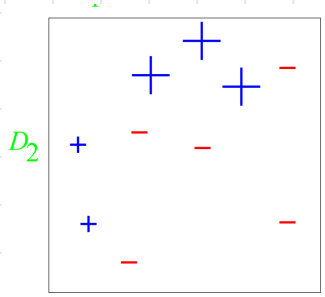
Q: What does it mean? What are the weights?

A: Consider the following example:

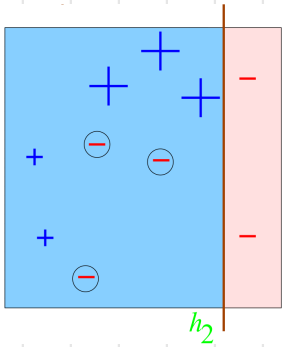


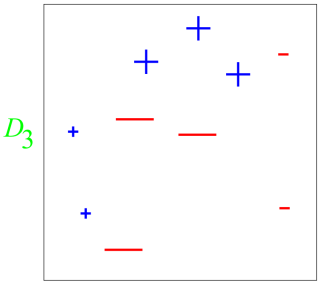
Round 1:





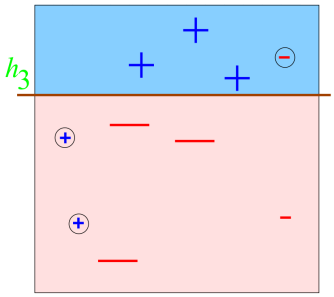
Round 2:



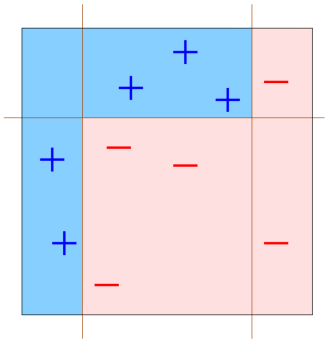


Wrongly classified samples are given more weights, so that in the next round (round 2), they will get more attention.

Round 3:



Final hypothesis:



→ a strong classifier, which is obtained by combining the weak classifiers (h_1, h_2, h_3)

[6] Training Algorithm based on AdaBoost

1. Input: descriptor & labels : $(x_1, y_1), \ldots (x_N, y_N)$

   x_i = a descriptor (160,000 length features) ; $N = 10{,}000$

   $y_i = \begin{cases} 0 & : \text{negative sample} \\ 1 & : \text{positive sample} \end{cases}$; i = the index of samples or descriptors

2. Initialize weights: $w_{1,i} = \begin{cases} 1/2m & \ominus \\ 1/2l & \oplus \end{cases}$; m, l = the numbers of negative & positive samples, respectively.

3. For $t = 1, \ldots, T$:

   (T = the number of the weak classifiers / dimensions of data)

   (1) Normalize the weights: $w_{t,i} = \dfrac{w_{t,i}}{\sum_{j=1}^{N} w_{t,j}}$

   (2) Select the best weak classifier: the one that minimizes the error:

   $$\epsilon_t = \min_{\{\theta\}} \sum_{i=1}^{N} w_{t,i} \, |h(x_i, \theta) - y_i|$$

   - ϵ_t: range : 0 ~ 1 (due to $w_{t,i}$)
   - $\min_{\{\theta\}}$: all functions / parameters of possible weak classifiers
   - $\sum_{i=1}^{N}$: every sample
   - $|h(x_i, \theta) - y_i|$: zero if x_i is correctly classified

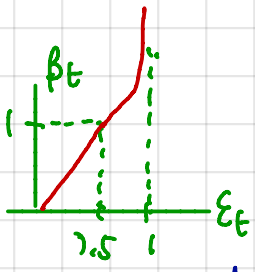
   (3) Define: $h_t(\vec{x}) = h(\vec{x}, \theta_t)$

   ↳ the best weak classifier that minimizes ϵ_t

   (either 0 or 1 (represents the weak classifier))

   (4) Update the weights: $w_{t+1,i} = w_{t,i} \, \beta_t^{(1-\alpha_i)}$

   where: $\alpha_i = \begin{cases} 0 & \text{if } x_i \text{ is correctly classified} \\ 1 & \text{else} \end{cases}$

   Cases:

   1. $\alpha_i = 1$ (incorrect classification) → w_{t+1} doesn't change
   2. $\alpha_i = 0$ (correct classification) & ϵ_t is small (< 0.5), w_{t+1} becomes smaller
   3. $\alpha_i = 0$ & $\epsilon_t > 0.5$, w_{t+1} becomes larger as $\beta_t > 1$ → to penalize & give x_i the same as case #1.

4. Strong classifier:

   $$C(x_{test}) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \gamma_t \, h_t(x_{test}) \geq \frac{1}{2} \sum_{t}^{T} \gamma_t \\ 0 & \text{else} \end{cases} \quad ; \quad \gamma_t = \log \frac{1}{\beta_t}$$

   - $\sum_{t=1}^{T}$: give more weight to a better h_t

[7] Testing: Simple Version

Input: a single image of size 24x24

Output: a label { 1 if face, 0 else }

Algorithm:

① Calculate the Haar-like descriptor from the input image

② Use AdaBoost & the trained parameters to decide/classify.

But, this is not what we want.

What we want: Given an image in any size, to know the locations of the faces (if any) and to draw bounding boxes surrounding those faces.

Naive Solution

Input: any image of any size (gray)

Algorithm

(1) Scan each pixel in the input image, and take a patch/subwindow with size of 24x24 pixels.

(2) Using the strong classifier, check if any of the subwindows is face.

(3) Increase the size of the subwindows i.e. 30x30 pixels, scan the entire pixels, and check again if any subwindows are faces.

(4) Keep increasing the size of the subwindows till a certain size (= max. face size)

Problem: SLOW

① Generating subwindows → Integral Image

② Checking using the strong classifier → Cascade

[9] Integral Image

1. Integral image formulation:

$$ii(x,y) = ii(x-1,y) + s(x,y) \qquad s(x,-1) = 0$$
$$s(x,y) = s(x,y-1) + i(x,y) \qquad ii(-1,y) = 0$$

$i(x,y)$: original input image

Example:

input image, i:

	0	1
0	5	2
1	3	6

ii: the integral image:

5	
8	16

$ii(0,1) = ii(-1,1) + s(0,1)$, $ii(-1,1) = 0$

$s(0,1) = s(0,0) + i(0,1)$, $i(0,1) = 3$

$s(0,0) = s(0,-1) + i(0,0) = 5$, $s(0,-1) = 0$, $i(0,0) = 5$

$ii(0,1) = 0 + 5 + 3 = 8$

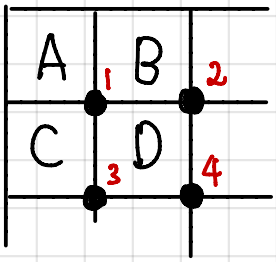
$ii(1,1) = ii(0,1) + s(1,1)$, $ii(0,1) = 8$

$s(1,1) = s(0,1) + i(1,1)$, $i(1,1) = 6$

$s(0,1) = s(1,-1) + i(1,0)$, $s(1,-1) = 0$, $i(1,0) = 2$

$ii(1,1) = 8 + 6 + 2 = 16$

2. Computing a block:



$1 = A$
$2 = A + B$
$3 = A + C$
$4 = A + B + C + D$

Given the values of 1, 2, 3, 4, what is the value of block D?

$$D = 4 - (A+B+C) = 4 - (2+C) = 4 - (2 + (3-A))$$
$$= 4 - (2+3-1) = (4+1) - (2+3)$$

Example:

i (input)

5	2	5	2
3	6	3	6
5	2	5	2
3	6	3	6

39 16

ii (integral image)

5	7	12	14
8	16	24	32
13	23	36	46
16	32	48	64

1 2 2 3 3 4

Total sum indicated by red points:

$D_{red} = (④ + ①) - (② + ③)$
$= (64 + 16) - (32 + 32) = 16$

Blue points:

$D_{blue} = (64 + 5) - (14 + 16) = 39$

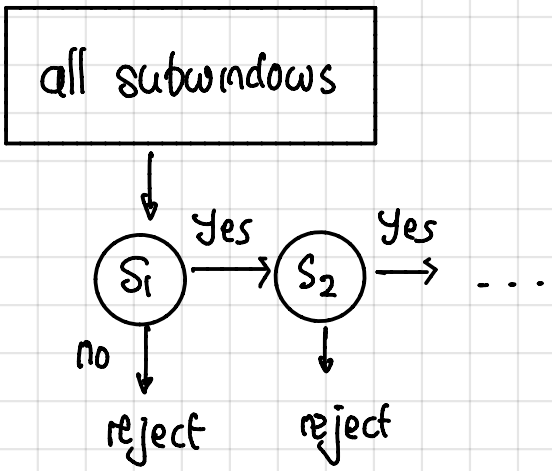
[10] Testing: Using Integral Image

Input: any gray image of any size

Algorithm:
(1) Compute the integral image
(2) Generate square windows starting from 24×24 pixels for all parts of the integral images
(3) Compute the descriptors using the standard mask sizes.
(4) Increase the sizes of the masks to capture larger square windows and compute the descriptor.

This will be similar to increasing the size of subwindows on the input image, but faster.

[11] Cascade of Classifiers



S_i = a stage composed of a few weak classifiers.

1. We want to reject soon, since non-face subwindows appear more frequently than face subwindows.
2. The number of the stages is determined by training the cascade.

Additional Notes:

1. $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$ (see page #6)

2. A strong classifier is defined as:

$$C(x_{test}) = \begin{cases} 1 & \sum_t^T \alpha_t h_t(x_{test}) \geq \frac{1}{2}\sum_t^T \alpha_t \\ 0 & \text{else} \end{cases}$$

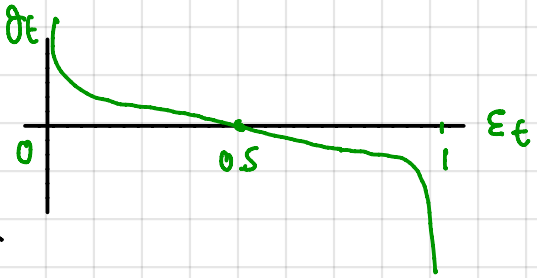
(1) Assuming $\alpha_t = 1$ for every t, then the condition is:

$$\sum_t^T h_t(x_{test}) \geq \frac{1}{2}T \quad ; \quad T = 160{,}000$$

This implies: For x_{test} to be labeled 1, at least it must be supported by 50% of the whole weak classifiers (up to T)

(2) $\alpha_t = \log\frac{1}{\beta_t} = \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) = \log\left(\frac{1}{\varepsilon_t} - 1\right)$

This implies:



if $\varepsilon_t \geq 0$ then $\alpha_t \leq 0$
else $\alpha_t > 0$

we only consider the weak classifiers whose error is small.

3. How to use $C(\bar{x})$, the strong classifier, in testing?

$$C(x_i, \bar{x}) = \begin{cases} 1 & \sum_t \alpha_t h_t(x_i, \bar{x}) > \frac{1}{2}\sum \alpha_t \\ 0 & \text{else} \end{cases}$$

where $h_t(x_i, \bar{x})$ means we apply the weak classifier h_t on x_i.

$C(x_i, \bar{x})$ means we label x_i to be 1 if most of the weak classifiers that have error less than 0.5, classify x_i as a positive sample.